FOUNDATION-0.2a: Core Database Schema - PART 1 (Code)

🎯 CONTEXT

Phase: FOUNDATION (Week 1 - Day 1 Morning)
Component: PostgreSQL Core Schema + Alembic Migration
Estimated Time: 30 min AI execution + 10 min verification
Complexity: MEDIUM
Risk Level: LOW (foundational work, well-tested patterns)
Files: Part 1 of 2 (Code implementation)

📦 DEPENDENCIES

Must Complete First:

- **PILOT-01:** Bootstrap project structure ✅ COMPLETED
- **PILOT-02:** Orchestrator skeleton ✅ COMPLETED
- **PILOT-03:** Cost Agent skeleton ✅ COMPLETED
- **PILOT-04:** LangGraph integration ✅ COMPLETED
- **PILOT-05:** Spot migration workflow ✅ COMPLETED

Required Services Running:


bash

```
# Verify infrastructure
cd ~/optiinfra
make verify

# Expected output:
# PostgreSQL... ✅ HEALTHY
# ClickHouse... ✅ HEALTHY
# Qdrant... ✅ HEALTHY
# Redis... ✅ HEALTHY

# Verify PostgreSQL connection
psql postgresql://optiinfra:password@localhost:5432/optiinfra -c "SELECT version();"

# Expected: PostgreSQL version output
```

Required Environment:

bash

```
# Verify Python environment
cd ~/optiinfra
python --version  # Python 3.11+

# Check if shared directory exists
ls shared/database/
# Should exist from PILOT-01 bootstrap
```

🎯 OBJECTIVE

Create a **complete PostgreSQL database schema** that provides the foundation for all OptiInfra agents. This schema will store:

1. **Customers** - Customer accounts and metadata
2. **Agents** - Registered agents (Cost, Performance, Resource, Application)
3. **Events** - System events and audit logs
4. **Recommendations** - Optimization recommendations from agents
5. **Approvals** - Customer approvals for recommendations
6. **Optimizations** - Executed optimizations and results

Success Criteria:

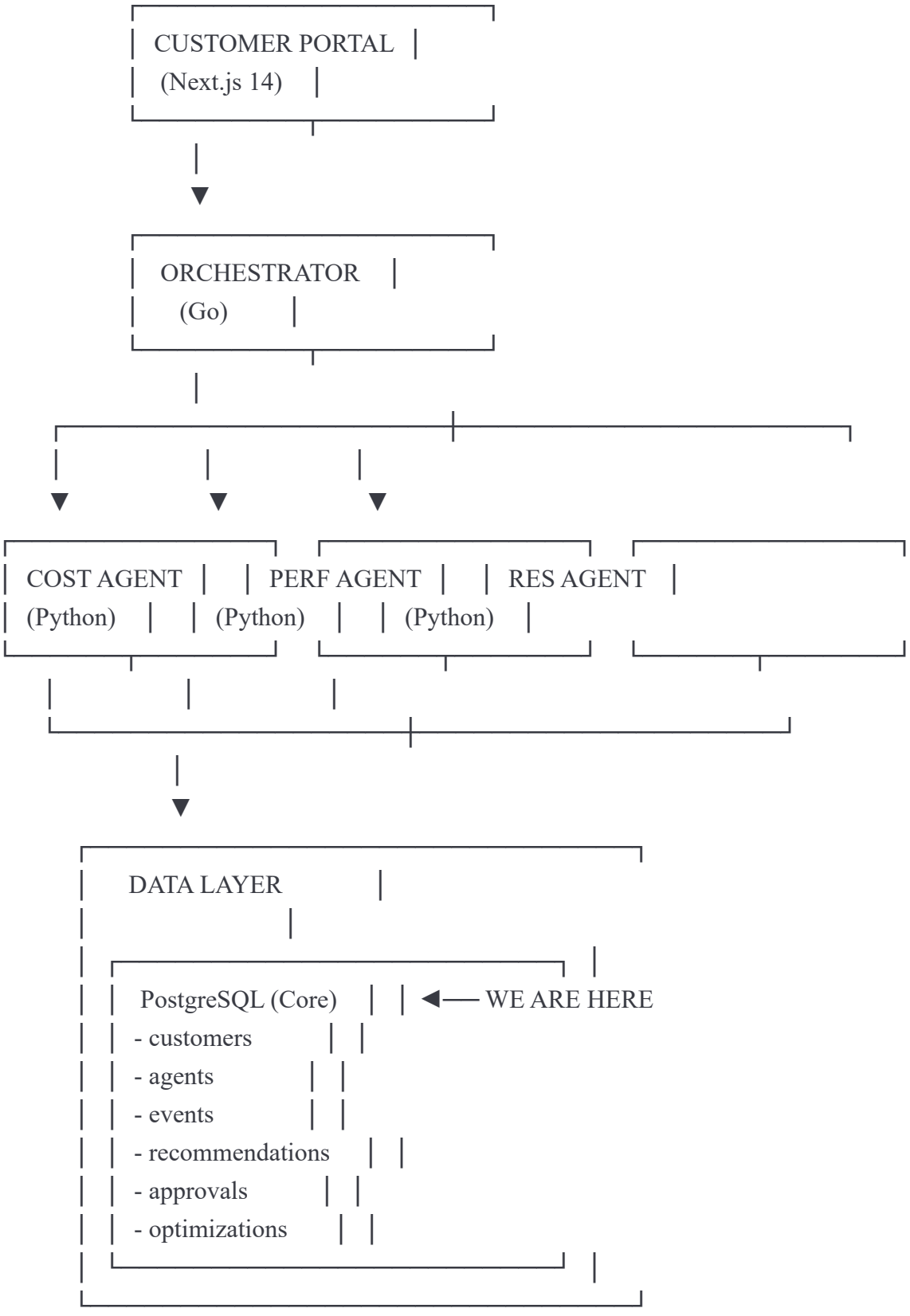
✅ Complete SQLAlchemy models created (6 tables)
✅ Alembic migration working (001_core_schema.py)
✅ Database tables created successfully
✅ Seed data inserted (test customers, agents) - **SEE PART 2**
✅ All relationships defined (foreign keys)
✅ Indexes created (for performance)
✅ Validation tests pass (15+ tests) - **SEE PART 2**
✅ Can query all tables successfully
✅ Documentation complete

Failure Signs:

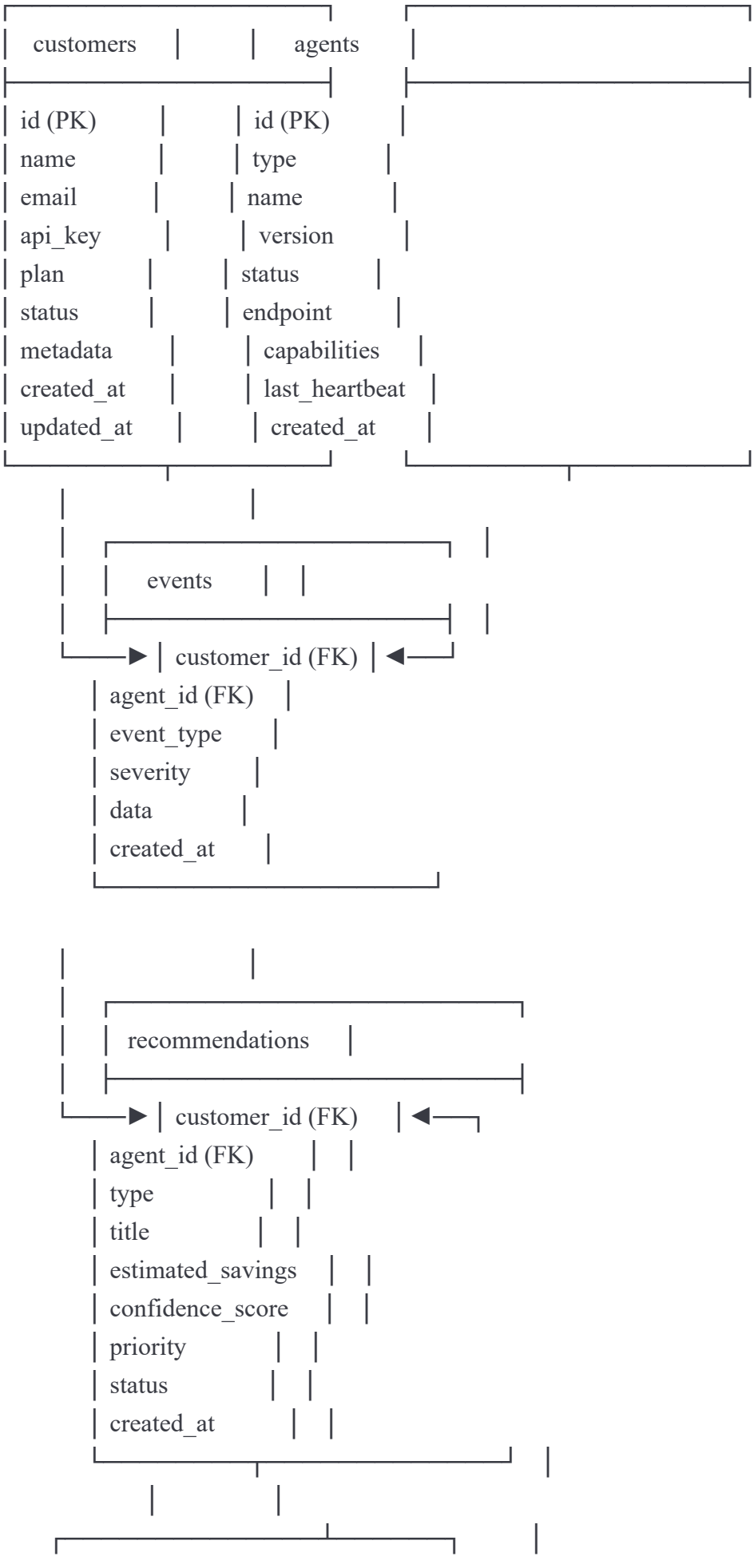
❌ Migration fails to run
❌ Tables not created
❌ Foreign key constraints broken
❌ Cannot insert test data
❌ Queries fail

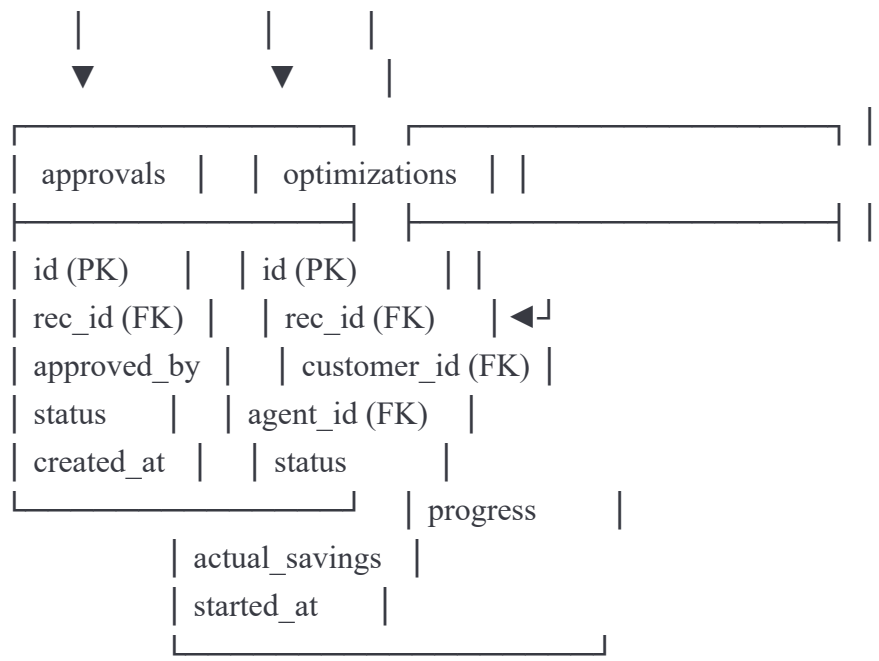
🗺️ TECHNICAL SPECIFICATION

Architecture Context



Database Schema Design


customers
id (PK)
name
email
api_key
plan
status
metadata
created_at
updated_at
agents
id (PK)
type
name
version
status
endpoint
capabilities
last_heartbeat
created_at
events
customer_id (FK)
agent_id (FK)
event_type
severity
data
created_at
recommendations
customer_id (FK)
agent_id (FK)
type
title
estimated_savings
confidence_score
priority
status
created_at



File Structure to Create (Part 1)

```
shared/
├── database/
│   ├── models/
│   │   ├── __init__.py        # MODIFY: Export core models ⭐
│   │   └── core.py            # CREATE: SQLAlchemy models ⭐
│   └── migrations/
│       └── versions/
│           └── 001_core_schema.py    # CREATE: Alembic migration ⭐
```

Note: Seeds, tests, and validation are in **PART 2**

💻 IMPLEMENTATION - PART 1

FILE 1: CREATE shared/database/models/core.py

This is the most important file - contains ALL SQLAlchemy models.

Copy this ENTIRE code to Windsurf:

python

```
"""
Core database models for OptiInfra.
Provides the foundation for all agents.
"""
import uuid
from datetime import datetime
from typing import Optional
from enum import Enum as PyEnum

from sqlalchemy import (
    Column, String, Integer, Float, Boolean, DateTime, Text,
    ForeignKey, Enum, Index, UniqueConstraint, CheckConstraint,
    DECIMAL, JSON
)
from sqlalchemy.dialects.postgresql import UUID, JSONB
from sqlalchemy.orm import relationship, declarative_base
from sqlalchemy.sql import func

# Base class for all models
Base = declarative_base()


# ============================================================================
# ENUMS
# ============================================================================

class CustomerPlan(str, PyEnum):
    """Customer subscription plans"""
    FREE = "free"
    STARTUP = "startup"
    ENTERPRISE = "enterprise"


class CustomerStatus(str, PyEnum):
    """Customer account status"""
    ACTIVE = "active"
    SUSPENDED = "suspended"
    CHURNED = "churned"


class AgentType(str, PyEnum):
```

```
    """Types of agents in the system"""
    ORCHESTRATOR = "orchestrator"
    COST = "cost"
    PERFORMANCE = "performance"
    RESOURCE = "resource"
    APPLICATION = "application"


class AgentStatus(str, PyEnum):
    """Agent health status"""
    STARTING = "starting"
    HEALTHY = "healthy"
    DEGRADED = "degraded"
    FAILED = "failed"
    STOPPED = "stopped"


class EventSeverity(str, PyEnum):
    """Event severity levels"""
    INFO = "info"
    WARNING = "warning"
    ERROR = "error"
    CRITICAL = "critical"


class RecommendationPriority(str, PyEnum):
    """Priority of recommendations"""
    LOW = "low"
    MEDIUM = "medium"
    HIGH = "high"
    CRITICAL = "critical"


class RecommendationStatus(str, PyEnum):
    """Status of recommendations"""
    PENDING = "pending"
    APPROVED = "approved"
    REJECTED = "rejected"
    EXECUTING = "executing"
    COMPLETED = "completed"
    FAILED = "failed"
```

```
    ROLLED_BACK = "rolled_back"


class ApprovalStatus(str, PyEnum):
    """Approval status"""
    PENDING = "pending"
    APPROVED = "approved"
    REJECTED = "rejected"


class OptimizationStatus(str, PyEnum):
    """Optimization execution status"""
    QUEUED = "queued"
    EXECUTING = "executing"
    COMPLETED = "completed"
    FAILED = "failed"
    ROLLED_BACK = "rolled_back"


# ============================================================================
# MODELS
# ============================================================================

class Customer(Base):
    """
    Customer accounts.

    Represents companies/users using OptiInfra.
    Each customer has multiple agents optimizing their infrastructure.
    """
    __tablename__ = "customers"

    id = Column(
        UUID(as_uuid=True),
        primary_key=True,
        default=uuid.uuid4,
        comment="Unique customer identifier"
    )
    name = Column(
        String(255),
        nullable=False,
```

```
    comment="Customer/company name"
)
email = Column(
    String(255),
    nullable=False,
    unique=True,
    index=True,
    comment="Primary contact email"
)
api_key = Column(
    String(64),
    nullable=False,
    unique=True,
    index=True,
    default=lambda: str(uuid.uuid4()).replace("-", ""),
    comment="API authentication key"
)
plan = Column(
    Enum(CustomerPlan),
    nullable=False,
    default=CustomerPlan.FREE,
    comment="Subscription plan"
)
status = Column(
    Enum(CustomerStatus),
    nullable=False,
    default=CustomerStatus.ACTIVE,
    index=True,
    comment="Account status"
)
metadata = Column(
    JSONB,
    nullable=True,
    default={},
    comment="Flexible metadata storage"
)
created_at = Column(
    DateTime(timezone=True),
    nullable=False,
    server_default=func.now(),
    comment="Account creation timestamp"
```

```
    )
    updated_at = Column(
        DateTime(timezone=True),
        nullable=False,
        server_default=func.now(),
        onupdate=func.now(),
        comment="Last update timestamp"
    )

    # Relationships
    events = relationship("Event", back_populates="customer", cascade="all, delete-orphan")
    recommendations = relationship("Recommendation", back_populates="customer", cascade="all, delete-orphan")
    optimizations = relationship("Optimization", back_populates="customer", cascade="all, delete-orphan")

    # Indexes
    __table_args__ = (
        Index("idx_customer_email", "email"),
        Index("idx_customer_api_key", "api_key"),
        Index("idx_customer_status", "status"),
    )

    def __repr__(self):
        return f"<Customer(id={self.id}, name='{self.name}', plan={self.plan.value})>"


class Agent(Base):
    """
    Registered agents in the system.

    Tracks all agents (Cost, Performance, Resource, Application, Orchestrator)
    with their capabilities, status, and health.
    """
    __tablename__ = "agents"

    id = Column(
        UUID(as_uuid=True),
        primary_key=True,
        default=uuid.uuid4,
        comment="Unique agent identifier"
    )
    type = Column(
```

```
    Enum(AgentType),
    nullable=False,
    index=True,
    comment="Agent type (cost, performance, etc.)"
)
name = Column(
    String(255),
    nullable=False,
    comment="Agent name/hostname"
)
version = Column(
    String(50),
    nullable=True,
    comment="Agent version (semver)"
)
status = Column(
    Enum(AgentStatus),
    nullable=False,
    default=AgentStatus.STARTING,
    index=True,
    comment="Current health status"
)
endpoint = Column(
    String(255),
    nullable=True,
    comment="Agent HTTP endpoint URL"
)
capabilities = Column(
    JSONB,
    nullable=False,
    default=[],
    comment="List of agent capabilities"
)
metadata = Column(
    JSONB,
    nullable=True,
    default={},
    comment="Additional agent metadata"
)
last_heartbeat = Column(
    DateTime(timezone=True),
```

```
        nullable=True,
        comment="Last heartbeat timestamp"
    )
    created_at = Column(
        DateTime(timezone=True),
        nullable=False,
        server_default=func.now(),
        comment="Agent registration timestamp"
    )
    updated_at = Column(
        DateTime(timezone=True),
        nullable=False,
        server_default=func.now(),
        onupdate=func.now(),
        comment="Last update timestamp"
    )

    # Relationships
    events = relationship("Event", back_populates="agent", cascade="all, delete-orphan")
    recommendations = relationship("Recommendation", back_populates="agent", cascade="all, delete-orphan")
    optimizations = relationship("Optimization", back_populates="agent", cascade="all, delete-orphan")

    # Indexes
    __table_args__ = (
        Index("idx_agent_type", "type"),
        Index("idx_agent_status", "status"),
        Index("idx_agent_heartbeat", "last_heartbeat"),
        UniqueConstraint("type", "name", name="uq_agent_type_name"),
    )

    def __repr__(self):
        return f"<Agent(id={self.id}, type={self.type.value}, name='{self.name}', status={self.status.value})>"


class Event(Base):
    """
    System events and audit logs.

    Tracks all significant events in the system for auditing,
    debugging, and analytics.
    """
```

```
__tablename__ = "events"

id = Column(
    UUID(as_uuid=True),
    primary_key=True,
    default=uuid.uuid4,
    comment="Unique event identifier"
)
customer_id = Column(
    UUID(as_uuid=True),
    ForeignKey("customers.id", ondelete="CASCADE"),
    nullable=False,
    index=True,
    comment="Associated customer"
)
agent_id = Column(
    UUID(as_uuid=True),
    ForeignKey("agents.id", ondelete="SET NULL"),
    nullable=True,
    index=True,
    comment="Agent that generated event (optional)"
)
event_type = Column(
    String(100),
    nullable=False,
    index=True,
    comment="Event type (e.g., optimization_started)"
)
severity = Column(
    Enum(EventSeverity),
    nullable=False,
    default=EventSeverity.INFO,
    index=True,
    comment="Event severity level"
)
data = Column(
    JSONB,
    nullable=False,
    default={},
    comment="Event payload/details"
)
```

```
    created_at = Column(
        DateTime(timezone=True),
        nullable=False,
        server_default=func.now(),
        index=True,
        comment="Event timestamp"
    )

    # Relationships
    customer = relationship("Customer", back_populates="events")
    agent = relationship("Agent", back_populates="events")

    # Indexes
    __table_args__ = (
        Index("idx_event_customer", "customer_id"),
        Index("idx_event_agent", "agent_id"),
        Index("idx_event_type", "event_type"),
        Index("idx_event_severity", "severity"),
        Index("idx_event_created", "created_at"),
        Index("idx_event_customer_created", "customer_id", "created_at"),
    )

    def __repr__(self):
        return f"<Event(id={self.id}, type='{self.event_type}', severity={self.severity.value})>"


class Recommendation(Base):
    """
    Optimization recommendations from agents.

    Agents analyze infrastructure and generate recommendations
    for cost savings, performance improvements, etc.
    """
    __tablename__ = "recommendations"

    id = Column(
        UUID(as_uuid=True),
        primary_key=True,
        default=uuid.uuid4,
        comment="Unique recommendation identifier"
    )
```

```
customer_id = Column(
    UUID(as_uuid=True),
    ForeignKey("customers.id", ondelete="CASCADE"),
    nullable=False,
    index=True,
    comment="Associated customer"
)
agent_id = Column(
    UUID(as_uuid=True),
    ForeignKey("agents.id", ondelete="CASCADE"),
    nullable=False,
    index=True,
    comment="Agent that created recommendation"
)
type = Column(
    String(100),
    nullable=False,
    index=True,
    comment="Recommendation type (spot_migration, etc.)"
)
title = Column(
    String(255),
    nullable=False,
    comment="Recommendation title"
)
description = Column(
    Text,
    nullable=True,
    comment="Detailed description"
)
estimated_savings = Column(
    DECIMAL(10, 2),
    nullable=True,
    comment="Estimated monthly savings (USD)"
)
estimated_improvement = Column(
    DECIMAL(10, 2),
    nullable=True,
    comment="Performance improvement multiplier"
)
confidence_score = Column(
```

```
    Float,
    nullable=False,
    default=0.0,
    comment="Confidence score (0.0 to 1.0)"
)
priority = Column(
    Enum(RecommendationPriority),
    nullable=False,
    default=RecommendationPriority.MEDIUM,
    index=True,
    comment="Recommendation priority"
)
status = Column(
    Enum(RecommendationStatus),
    nullable=False,
    default=RecommendationStatus.PENDING,
    index=True,
    comment="Current status"
)
data = Column(
    JSONB,
    nullable=False,
    default={},
    comment="Recommendation details/metadata"
)
created_at = Column(
    DateTime(timezone=True),
    nullable=False,
    server_default=func.now(),
    index=True,
    comment="Creation timestamp"
)
updated_at = Column(
    DateTime(timezone=True),
    nullable=False,
    server_default=func.now(),
    onupdate=func.now(),
    comment="Last update timestamp"
)
approved_at = Column(
    DateTime(timezone=True),
```

```
        nullable=True,
        comment="Approval timestamp"
    )
    executed_at = Column(
        DateTime(timezone=True),
        nullable=True,
        comment="Execution start timestamp"
    )

    # Relationships
    customer = relationship("Customer", back_populates="recommendations")
    agent = relationship("Agent", back_populates="recommendations")
    approvals = relationship("Approval", back_populates="recommendation", cascade="all, delete-orphan")
    optimizations = relationship("Optimization", back_populates="recommendation", cascade="all, delete-orphan")

    # Indexes and Constraints
    __table_args__ = (
        Index("idx_rec_customer", "customer_id"),
        Index("idx_rec_agent", "agent_id"),
        Index("idx_rec_type", "type"),
        Index("idx_rec_priority", "priority"),
        Index("idx_rec_status", "status"),
        Index("idx_rec_created", "created_at"),
        Index("idx_rec_customer_status", "customer_id", "status"),
        CheckConstraint("confidence_score >= 0.0 AND confidence_score <= 1.0", name="check_confidence_range"),
    )

    def __repr__(self):
        return f"<Recommendation(id={self.id}, type='{self.type}', status={self.status.value}, savings=${self.estimated_savi


class Approval(Base):
    """
    Customer approvals for recommendations.

    Tracks which recommendations customers have approved/rejected.
    """
    __tablename__ = "approvals"

    id = Column(
        UUID(as_uuid=True),
```

```
    primary_key=True,
    default=uuid.uuid4,
    comment="Unique approval identifier"
)
recommendation_id = Column(
    UUID(as_uuid=True),
    ForeignKey("recommendations.id", ondelete="CASCADE"),
    nullable=False,
    index=True,
    comment="Associated recommendation"
)
approved_by = Column(
    String(255),
    nullable=False,
    comment="User who approved/rejected (email or ID)"
)
status = Column(
    Enum(ApprovalStatus),
    nullable=False,
    default=ApprovalStatus.PENDING,
    index=True,
    comment="Approval status"
)
comment = Column(
    Text,
    nullable=True,
    comment="Optional approval comment"
)
created_at = Column(
    DateTime(timezone=True),
    nullable=False,
    server_default=func.now(),
    comment="Approval request timestamp"
)
updated_at = Column(
    DateTime(timezone=True),
    nullable=False,
    server_default=func.now(),
    onupdate=func.now(),
    comment="Last update timestamp"
)
```

```
    # Relationships
    recommendation = relationship("Recommendation", back_populates="approvals")

    # Indexes
    __table_args__ = (
        Index("idx_approval_rec", "recommendation_id"),
        Index("idx_approval_status", "status"),
    )

    def __repr__(self):
        return f"<Approval(id={self.id}, rec_id={self.recommendation_id}, status={self.status.value})>"


class Optimization(Base):
    """
    Executed optimizations and their results.

    Tracks the execution of approved recommendations,
    including progress, results, and actual impact.
    """
    __tablename__ = "optimizations"

    id = Column(
        UUID(as_uuid=True),
        primary_key=True,
        default=uuid.uuid4,
        comment="Unique optimization identifier"
    )
    recommendation_id = Column(
        UUID(as_uuid=True),
        ForeignKey("recommendations.id", ondelete="CASCADE"),
        nullable=False,
        index=True,
        comment="Associated recommendation"
    )
    customer_id = Column(
        UUID(as_uuid=True),
        ForeignKey("customers.id", ondelete="CASCADE"),
        nullable=False,
        index=True,
```

```
    comment="Associated customer"
)
agent_id = Column(
    UUID(as_uuid=True),
    ForeignKey("agents.id", ondelete="CASCADE"),
    nullable=False,
    index=True,
    comment="Agent executing optimization"
)
status = Column(
    Enum(OptimizationStatus),
    nullable=False,
    default=OptimizationStatus.QUEUED,
    index=True,
    comment="Execution status"
)
progress = Column(
    Integer,
    nullable=False,
    default=0,
    comment="Progress percentage (0-100)"
)
result = Column(
    JSONB,
    nullable=True,
    default={},
    comment="Execution results/details"
)
error = Column(
    Text,
    nullable=True,
    comment="Error message if failed"
)
actual_savings = Column(
    DECIMAL(10, 2),
    nullable=True,
    comment="Actual monthly savings achieved (USD)"
)
actual_improvement = Column(
    DECIMAL(10, 2),
    nullable=True,
```

```
    comment="Actual performance improvement"
)
started_at = Column(
    DateTime(timezone=True),
    nullable=True,
    comment="Execution start timestamp"
)
completed_at = Column(
    DateTime(timezone=True),
    nullable=True,
    comment="Execution completion timestamp"
)
rolled_back_at = Column(
    DateTime(timezone=True),
    nullable=True,
    comment="Rollback timestamp (if applicable)"
)
created_at = Column(
    DateTime(timezone=True),
    nullable=False,
    server_default=func.now(),
    comment="Creation timestamp"
)

# Relationships
recommendation = relationship("Recommendation", back_populates="optimizations")
customer = relationship("Customer", back_populates="optimizations")
agent = relationship("Agent", back_populates="optimizations")

# Indexes and Constraints
__table_args__ = (
    Index("idx_opt_rec", "recommendation_id"),
    Index("idx_opt_customer", "customer_id"),
    Index("idx_opt_agent", "agent_id"),
    Index("idx_opt_status", "status"),
    Index("idx_opt_customer_status", "customer_id", "status"),
    CheckConstraint("progress >= 0 AND progress <= 100", name="check_progress_range"),
)
```

```python
    def __repr__(self):
        return f"<Optimization(id={self.id}, status={self.status.value}, progress={self.progress}%)>"
```

✅ **FILE 1 COMPLETE (~400 lines, ~14,000 characters)**

FILE 2: MODIFY shared/database/models/init.py

Update the package exports to include all core models:

python

```
"""
Database models package.
"""
from shared.database.models.core import (
    Base,
    # Enums
    CustomerPlan,
    CustomerStatus,
    AgentType,
    AgentStatus,
    EventSeverity,
    RecommendationPriority,
    RecommendationStatus,
    ApprovalStatus,
    OptimizationStatus,
    # Models
    Customer,
    Agent,
    Event,
    Recommendation,
    Approval,
    Optimization,
)

__all__ = [
    "Base",
    # Enums
    "CustomerPlan",
    "CustomerStatus",
    "AgentType",
    "AgentStatus",
    "EventSeverity",
    "RecommendationPriority",
    "RecommendationStatus",
    "ApprovalStatus",
    "OptimizationStatus",
    # Models
    "Customer",
    "Agent",
    "Event",
    "Recommendation",
```

```
    "Approval",
    "Optimization",
]
```

✅ **FILE 2 COMPLETE**

FILE 3: CREATE shared/database/migrations/versions/001_core_schema.py

Alembic migration to create all tables in PostgreSQL:

python

```
"""
Create core schema tables

Revision ID: 001_core_schema
Revises:
Create Date: 2025-10-18 12:00:00.000000
"""
from alembic import op
import sqlalchemy as sa
from sqlalchemy.dialects import postgresql

# revision identifiers, used by Alembic.
revision = '001_core_schema'
down_revision = None
branch_labels = None
depends_on = None


def upgrade():
    """Create core schema tables"""

    # Create ENUM types
    op.execute("""
        CREATE TYPE customerplan AS ENUM ('free', 'startup', 'enterprise');
        CREATE TYPE customerstatus AS ENUM ('active', 'suspended', 'churned');
        CREATE TYPE agenttype AS ENUM ('orchestrator', 'cost', 'performance', 'resource', 'application');
        CREATE TYPE agentstatus AS ENUM ('starting', 'healthy', 'degraded', 'failed', 'stopped');
        CREATE TYPE eventseverity AS ENUM ('info', 'warning', 'error', 'critical');
        CREATE TYPE recommendationpriority AS ENUM ('low', 'medium', 'high', 'critical');
        CREATE TYPE recommendationstatus AS ENUM ('pending', 'approved', 'rejected', 'executing', 'completed', 'failed', 'r
        CREATE TYPE approvalstatus AS ENUM ('pending', 'approved', 'rejected');
        CREATE TYPE optimizationstatus AS ENUM ('queued', 'executing', 'completed', 'failed', 'rolled_back');
    """)

    # Create customers table
    op.create_table(
        'customers',
        sa.Column('id', postgresql.UUID(as_uuid=True), primary_key=True),
        sa.Column('name', sa.String(255), nullable=False),
        sa.Column('email', sa.String(255), nullable=False, unique=True),
        sa.Column('api_key', sa.String(64), nullable=False, unique=True),
```

```
sa.Column('plan', postgresql.ENUM(name='customerplan'), nullable=False, server_default='free'),
sa.Column('status', postgresql.ENUM(name='customerstatus'), nullable=False, server_default='active'),
sa.Column('metadata', postgresql.JSONB, nullable=True, server_default='{}
```